
LABORATÓRIO 29

TEMPLATES E SOBRECARGA DE FUNÇÕES

EXERCÍCIOS DE FIXAÇÃO

VOCÊ DEVE FAZER OS EXERCÍCIOS PARA FIXAR O CONTEÚDO

1. Escreva uma função que normalmente recebe um argumento, o endereço de uma string, e exibe essa string uma vez. No entanto, se um segundo argumento, de tipo booleano, for fornecido e possuir valor verdadeiro, a função deve mostrar a string um número de vezes igual ao número de chamadas da função.
2. Considerando o modelo de registro abaixo:

```
struct Guloseima
{
    char marca[24];
    double peso;
    int calorias;
};
```

- a. Escreva uma função que receba uma referência para um registro **Guloseima**, um ponteiro para char, um double e um int e use os três últimos argumentos como valores do registro. Os últimos parâmetros devem ter os valores padrão "Charge", 40.0, 187.
 - b. Escreva uma função que receba uma referência para um registro **Guloseima** e exiba o seu conteúdo na tela. Use const sempre que for apropriado.
3. Usando o código abaixo como ponto de partida, complete o programa fornecendo as funções descritas. Observe que devem existir duas funções mostrar, uma para o tipo String e outra para constantes strings, cada uma usando argumentos padrão. Use const nos parâmetros quando for apropriado. O operador new deve ser usado para alocar memória suficiente para guardar o texto no registro String.

```

#include <iostream>
#include <cstring>
using namespace std;

struct String
{
    char * str;    // ponteiro para a string
    int  comp;    // comprimento da string (sem contar '\0')
};

// protótipos para ajustar(), mostrar() e mostrar()

int main()
{
    String msg;
    char teste[] = "Realidade de ponteiros e strings\n";

    ajustar(msg, teste); // primeiro argumento é uma referência
                        // aloca espaço para guardar cópia de teste
                        // ajusta o membro str de msg para apontar
                        // para novo bloco, copia teste para o novo
                        // bloco e ajusta o membro comp

    mostrar(msg);        // mostra o membro String uma vez
    mostrar(msg, 2);     // mostra o membro String duas vezes
    teste[0] = 'D';
    teste[1] = 'u';
    mostrar(teste);      // mostra a string uma vez
    mostrar(teste, 3);   // mostra a string três vezes
    mostrar("Pronto!");

    delete[] msg.str;
    return 0;
}

```

4. Construa uma função template `max5()` que receba um vetor de 5 elementos de tipo `T` e retorne o maior elemento do vetor. Como o tamanho do vetor é fixo, não há necessidade de receber o tamanho do vetor por parâmetro. Teste a função em um programa que crie um vetor de 5 inteiros e outro de 5 doubles.
5. Construa uma função template `maxn()` que receba um vetor de elementos de tipo `T` e um inteiro representando o número de elementos no vetor e retorne o maior elemento do vetor. Teste a função em um programa que crie um vetor de 6 inteiros e outro de 4 doubles.

EXERCÍCIOS DE APRENDIZAGEM

VOCÊ DEVE ESCREVER PROGRAMAS PARA REALMENTE APRENDER

1. Construa uma função que leia um certo número de linhas do arquivo texto exibido abaixo e mostre um sumário das informações. Em cada linha existem 4 valores: o número do laboratório, a quantidade de exercícios de revisão, a quantidade de exercícios de fixação e a quantidade de exercícios de aprendizagem.

```
0 0 0 7
1 2 8 4
2 6 4 5
3 2 3 6
4 2 3 6
5 2 3 7
6 1 3 5
7 0 5 5
8 0 5 4
9 1 4 5
10 0 3 4
11 0 5 5
12 2 4 6
13 1 2 4
14 5 4 4
15 4 5 5
16 4 2 4
17 4 3 5
18 4 3 6
19 2 4 5
20 5 4 4
21 5 4 7
22 2 5 5
23 2 5 3
24 3 4 3
25 3 3 4
26 3 3 5
27 2 3 3
28 1 4 3
29 0 5 3
```

```
-----
1a unidade
-----
Revisão: 16
Fixação: 38
Aprendi: 54
-----
Total: 108

-----
2a unidade
-----
Revisão: 26
Fixação: 35
Aprendi: 48
-----
Total: 109

-----
3a unidade
-----
Revisão: 26
Fixação: 40
Aprendi: 40
-----
Total: 106
```

A função deve receber uma referência para um objeto do tipo ifstream, o número da unidade e a quantidade de linhas que devem ser lidas. Use um argumento padrão igual a 10 para a quantidade de linhas. A função deve funcionar com o código abaixo:

```
int main() {
    ifstream fin;
    fin.open("stats.txt");
    unidade(fin, 1);
    unidade(fin, 2);
    unidade(fin, 3);
    fin.close();
}
```

2. Escreva funções sobrecarregadas para exibir números de tipo inteiro, números de tipo ponto-flutuante e constantes strings. Cada tipo deve ser exibido com uma cor diferente, de forma que o código abaixo gere a saída a seguir:

```
int main()
{
    cout << "Inteiro: ";
    print(45);
    cout << "\nPonto-flutuante: ";
    print(3.9);
    cout << "\nString: ";
    print("Oi Mundo");
    cout << "\n";
}
```

```
Inteiro: 45
Ponto-flutuante: 3.9
String: Oi Mundo
```

3. A função print exibida abaixo funciona apenas para valores de tipos inteiros. Se tentarmos chamá-la com valores ponto-flutuante, caractere ou string, teremos o seguinte resultado:

```
#include <iostream>
#include <string>
using namespace std;

void print(int a, int b)
{
    int c = a + b;
    cout << a << " + " << b << " = " << c << endl;
}

int main()
{
    string strA = "Pro";
    string strB = "gramando";
    print(1, 2);
    print(2.6, 3.7);
    print('A', 'B');
    //print(strA, strB);
}
```

```
1 + 2 = 3
2 + 3 = 5
65 + 66 = 131
// erro
```

É possível corrigir todos esses problemas usando programação genérica com templates. Transforme a função print em um template, de forma que ela trate corretamente todos os tipos de dados do programa.